Algorithm to create tree of possible system states during its functioning according to the given network of operations*

Abstract. We consider the system's functioning such that this functioning is fulfilled according to the given network of operations (i.e., according to the provided plan). The results of operations in the network and thus system functioning results are characterized by random values. To systematize possible states during the network of operations, we propose a graph-theoretic model—a tree of possible states. Each node of the tree corresponds to the system's workplace state, and each branch to the possible system state. Each possible workplace state corresponds to the possible operation or waiting for the operation's realization at this workplace. Thus, the tree of possible states corresponds to the tree of the network of operations' possible cuts. To create trees of possible states, an algorithm to build such trees from networks of operations is suggested. It differs from a known algorithm of maximum network flow in that all possible cuts of the network and all possible states are modeled. It differs from a known algorithm of building all antichains of the partially ordered set (poset) in that the network of operations we consider may not constitute such poset.

Key words. Network, Graph, Tree, Operations, Probabilistic, Estimation

AMS subject classifications. 68R10, 90B15, 90B90

1. Introduction.

In many areas of research [1][8], possible states of system functioning must be modeled. Such states are determined by operations (actions) and waiting at system workplaces performed simultaneously or sequentially. These states are random because the operations' duration and other characteristics of activities are random. States must be estimated to assess the random resources spent at each moment or to estimate random states' characteristics such that these states may result in functioning alternations [4][**?**]. To compute such states' probabilities and characteristics, we propose a model [3][**?**] of system state formation based on workplaces' states. It has the form of a tree where each node, excluding the root, corresponds to one of a system's workplace states during one of the operations or waiting. Each branch of the tree corresponds to one possible state of the system at some moment during operations fulfillment according to the given network. The root corresponds to all possible states during system functioning according to the given network of operations. Thus, the network of operations corresponds to the tree and vice versa. To build such a tree algorithm, it is necessary to build a tree of possible cuts of the network. This algorithm is discussed in this paper with a few variations illustrated by examples and visual representations of networks and trees built. Variations of the algorithm determined by networks of operations' representations are used as initial data for building trees of network cuts. Unlike in minimum network cut/maximum network flow algorithms, flow is not transferred between vertices or edges. Instead, each state corresponds to each cut of the network considered. Such a state is usually different for each set of vertices in the cut. Additionally, each cut of the network (and corresponding state) is random, and its probability is computed based on the given moment and operations' duration in the network. The probabilities of cuts form a probability space

*Submitted to the editors DATE.
Funding:

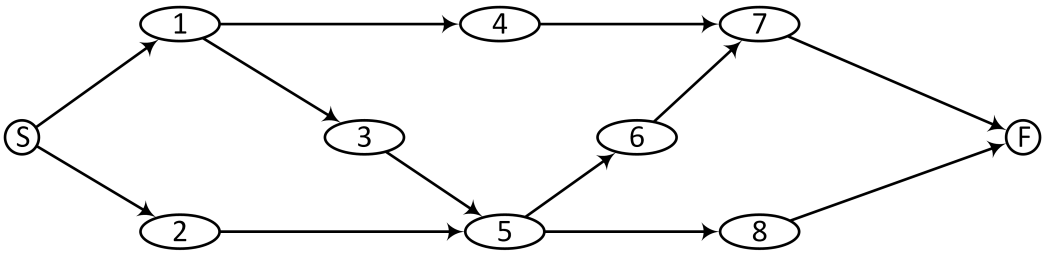

Figure 1. *The Network of Operations*

(of possible cuts of the network and thus the states of the system) at each moment. Thus, all possible cuts shall be constructed, computed depending on the moment, and considered in further research. Each cut corresponds to operations and waits, which can be fulfilled at one given moment (under unspecified parameters of operations). That is, such operations (waits) are not ordered by the network of operations. A similar need to build all possible sets of "incomparable" elements exists for posets, where they are named "anti-chains." Networks of operations are not necessarily partially ordered sets. The model and algorithm suggested for use in project management and business process planning applications where functioning can be interrupted due to environmental impacts and changes. Possible states of interruption and alternation can be modeled with the use of possible network cuts. Interaction of the environment and the system is subject to the research at the theory of games with the environment [5]. Game theory used for organization design [10]. But, modeling of system reaction on environmental impacts as a set of alternating networks of actions is not known to us. Such networks of actions are a natural way to represent organizational systems functioning, for example, to represent complex innovation projects [4]. Control of dynamic systems deals with the system control and reactions to the environmental impact [9]. There are discrete events dynamic models of control [6], networks of objects control [7] models and methods, Boolean networks control models and methods [2], but the control of alternating action networks is not known too. The paper is organized as follows. Our main results are in section 2, section 3, section 4, our new algorithm illustrated in section 4 and the conclusions follow in section 6.

2. The Possible Inputs of the Algorithm. We use networks of operations with a start and finish vertices. The network of operations is Directed Acyclic Graph (DAG), such that each vertex associated with operations on certain workplaces of the system. Start vertex associated with the operation of waiting to start at the required moment. It has no incoming edges. Finish operation is the operation of waiting to report the results of actions. Networks can be obtained with the use of process mining techniques [11]. Example of the network of operations shown in Figure 1. There are eight operations, operations 1-8. The goal of our algorithm is to build all possible cuts of the network. All possible cuts can be represented as all maximal cuts and their subsets. Such maximum cuts of the given network are schematically shown in Figure 2 with dotted lines. Each dotted line corresponds to a maximal cut. Its subsets may be cuts as well - due to possible waiting operations. It is possible to include waiting in the network explicitly. In this way, possible waiting included into any incoming edge if there are more than one incoming edges which lead to the same ending operation -

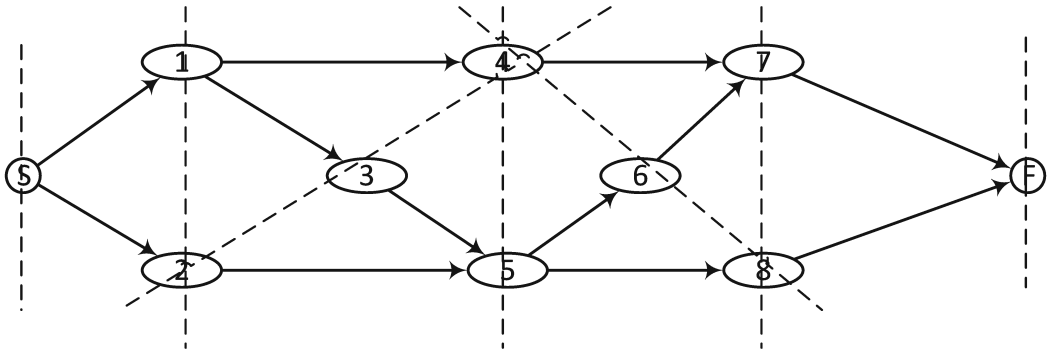

Figure 2. *Maximal Cuts of the Network of Operation*

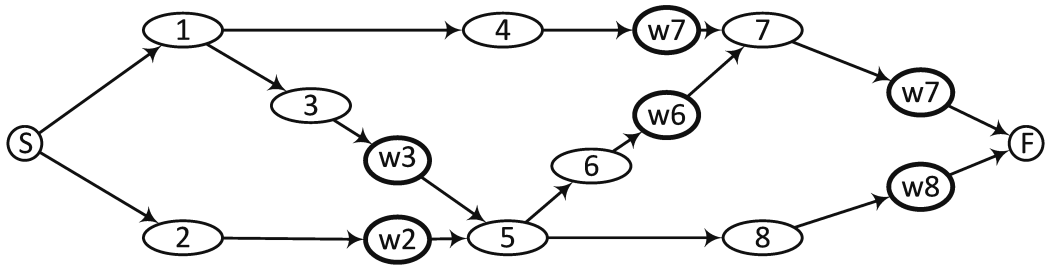

Figure 3. *The Network of Operations with Added Waiting Operations*

as in Figure 3 Such waiting operations appear as a result of modeling assumptions, which is: once operation may start, it will start without any waiting. Thus, if only one edge is in the input set of the vertex in the initial network, this edge can not split with waiting and vice versa. Alternatively, each subset of any maximal cut can be considered, but the probabilities of some subsets realization can be 0 by default. These probabilities counted based on the same assumption, but in functional form: "once operation may start, it will start without waiting time equal to 0". So, the probability of such waiting to be realized will always be 0. Under other assumptions, such waiting and the corresponding subset of maximum cut may be possible, All three cases of input networks require the same basic algorithm to build the tree of network cuts: an algorithm to build the tree of maximal cuts.

3. The Structure of the Algorithm. The algorithm illustrated in Figures 4-11. Each figure shows changes taking place at its steps. The algorithm consists of **(1)** selecting working (sub) routes at network according to Depth First Search (DFS) step of forward traverse and adding them to (sub) routes list, **(2)** constructing tree branches (i.e., parts of cuts) using working (sub) routes lists selected, **(3)** saving and rewriting working routes list. The algorithm creates and updates lists of active routes during the traverse. Route created when new vertices traversed. Routes deleted on the way forward in the network when vertex marked visited, and this vertex has few incoming routes, which are ones to delete. Routes can be restored when network traversed back through vertex were routes where deleted. Tree nodes added according to active routes. Thus, algorithm link traversed vertices, routes, and tree nodes.

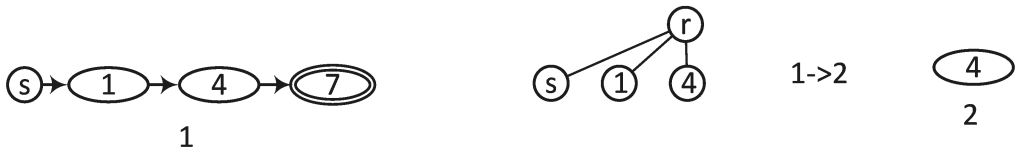

Figure 4. *Step 1*

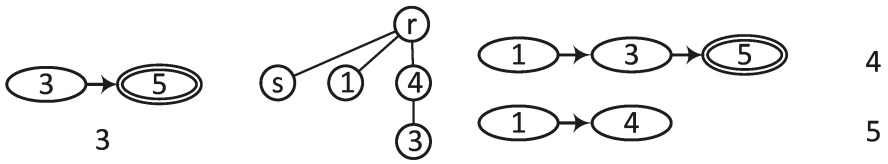

Figure 5. *Step 2*

Steps of algorithm performed sequentially, while traversing network and shown as left **(1)**, center **(2)** and right **(3)** parts in algorithm steps illustration in figures 4-11. The initial state of the algorithm is **(1)** start (s) vertex, corresponding to its root (r) node of the tree constructed **(2)** and start as the initial route in the list of routes **(3)**. The first step of the algorithm shown in Figure 4 and the last one in Figure 11. The last step returns the tree of network cuts constructed.

4. Example of The Algorithm Realization. The algorithm illustrated with the use of the network of operation in Figure 1. We illustrate maximal cuts tree construction. The result of the algorithm conforms to all cuts of the network shown in Figure 2. In figure 4 first step of the algorithm example illustrated.

(1.1): Traverse to the depth of the network until possible, according to DFS. The initial route set to route (s,1,4 - number 1 route in a list) as a result according to DFS traverse to the depth, vertex 7 marked as not yet reachable but edge (4,7) marked as visited. Choosing between multiple routes may be random.

(1.2): Nodes (s,1,4) are created in the tree as the first level of the tree - next to the root. The nodes of the tree are the result of mapping from route vertices to nodes.

(1.3): Return to vertex (1) according to DFS step backward to number 1 vertex and so number 1 route replaced with number 2 route, which consists of the vertex (4).

Proceed to the next step. In figure 5 second step illustrated.

(2.1): Traverse in-depth with DFS. The choice of vertex visits may be random. In our case, it is one possible choice of (3) which becomes route 3. Vertex (5) marked as unreachable yet, but incoming edge (3,5) visited. New routes added to saved one, so routes (2) and (3) are in the list of routes.

(2.2): Node (3) is created in the tree as the second level of the tree and connected to node 4 of route 2.

(2.3): Return back to vertex (s) according DFS. Routes (2) and (3) changed (restored) to (4) and (5) by adding vertex (1) visited during traversing backward to s. As a result, (1) added to heads of both routes.

Algorithm 4.1 DFS

```
Define
verts = ['0','1','2','3','4','5','6','7','8','9']
first=0
last=9
edges = [(0,1),(0,2),(1,4),(1,3),(2,5),(4,7),(5,6),(5,8),(6,7),(7,9),(8,9)]
adjlist :
[
0 [1,2]
1 [3,4]
2 [5]
3 [5]
4 [7]
5 [6,8]
7 [9]
8 [9]
]
VisitedVertex[NULL]
stack[0]
while stack do
  current=stack.pop
  for Neighbor in adjlist[current] do
    if not Neighbor in VisitedVertex then
      stack.append(Neighbor)
    end if
    VisitedVertex.append(current)
  end for
end while
return VisitedVertex
return tree
```

Proceed to the next step. In figure 6 third step illustrated.

(3.1): Proceed in-depth with DFS. Route 6 created with vertex (2) according to DFS, vertex five marked as reachable but not yet processed. New route added to saved ones, so routes (4), (5), and (6) active.

(3.2): Node (2) is created in the tree twice. First, as the third level of the tree and connected to node one on the third level. The second node added as a child to the node (3) at the fourth level of the tree. Node (3) is chosen because it is dangling and precedes by node (4), which is in the list of routes.

(3.3): According DFS vertex (5) marked for processing. As a result, routes (5) and (6), which ends in the vertex (5) deleted. Route (4) changed to route (2) due to vertex (1) removed from it during vertex (5) preparation for processing.

Proceed to the next step. In figure 7 fourth step illustrated.

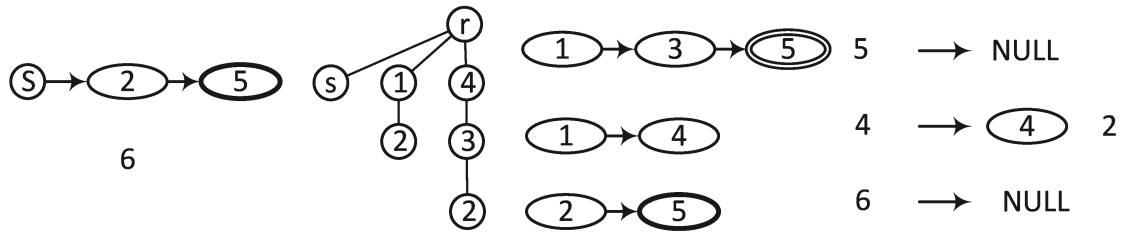

Figure 6. *Step 3*

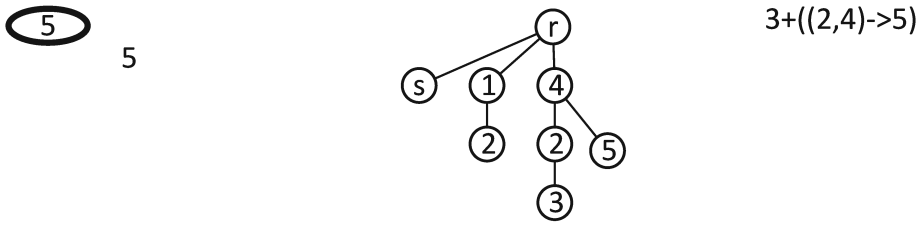

Figure 7. *Step 4*

(4.1): Visit vertex (5). Route number 7 created. As a result, vertex 5 marked as processed. New route added to saved ones, so routes (2) and (7) active.

(4.2): Node (5) is created in the tree as the third level of the tree and connected to node 4 (taken from route 2) on the second level.

(4.3): Vertex (5) marked as starting for further DFS and route (7) deleted from the list.

Proceed to the next step. In figure 8 fifth step illustrated.

(5.1): Traverse forward, according to DFS. Visit sequence may be random, but we choose (6,7) vertices. Route number 8 is created, which includes vertex (6) Vertex (7) marked as possible to visit. The new route (8) added to saved ones, so routes (2) and (8) active.

(5.2): Node (6) is created in the tree as the third level of the tree and connected to node 4 (taken from route 2) on the second level.

(5.3): Vertex (7) prepared for processing. Thus (2) and (8) routes deleted.

Proceed to the next step.

In figure 8 sixth step illustrated.

(6.1): Traverse forward, according to DFS. Visit (7). Route number 9 created, which includes vertex (7) Vertex (7) marked as visited and start for further DFS. The new route (9) added to saved ones (empty list).

(6.2): Node (7) is created in the tree as the second level of the tree and connected to root.

(6.3): No routes changed.

Proceed to the next step.

In figure 10 seventh step illustrated.

(7.1): Traverse forward, according to DFS. Because it is impossible to visit (F) return to vertex (5), according to DFS. Corresponding routes (2) (8) restored. Than traverse forward,

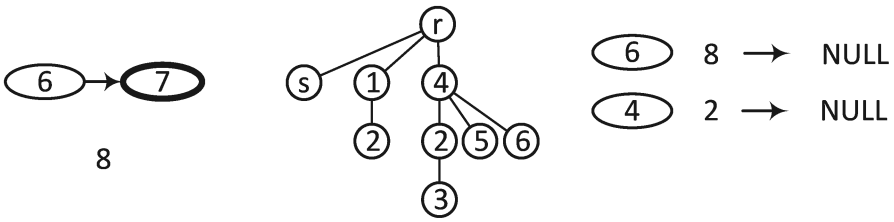

Figure 8. *Step 5*

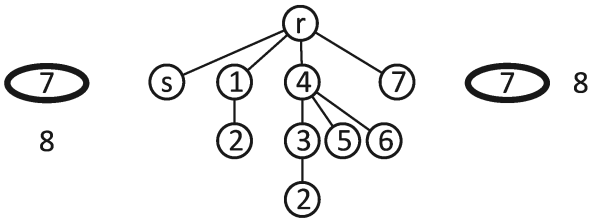

Figure 9. *Step 6*

according DFS, visit vertex (8). Route number (11) created and added to the list of routes, so the list of routes is (2), (8), (11).

(7.2): Node (8) is created in the tree as the fourth level of the tree and connected to dangling vertex (6), which preceded by node (6) and (4).

(7.3): Return to (7) back, so delete routes (2) and (8) and restore (10). So, the list of routes is (10, 11).

Proceed to the next step.

In figure 11 last (eighth, ninth and finish) steps illustrated.

(8.1): Traverse forward, according to DFS. Mark (F) as possible to visit.

(8.2): Node (8) is created in the tree as the third level of the tree and connected to vertex (7).

(8.3): No changes to the list.

Proceed to the next step.

(9.1): Process (F), delete routes (8) and (11). Create (last) route (12), which consists of (F) vertex.

(9.2): Node (F) is created in the tree as the second level of the tree and connected to root.

(9.3): No changes to the list.

(10.1): Try DFS from (F). Finish.

(10.2): No changes to the tree. Return Tree.

(10.3): Delete list of routes with the route (12) in it.

The tree of cuts obtained as a result corresponds to network cuts constructed manually and shown in Figure 2.

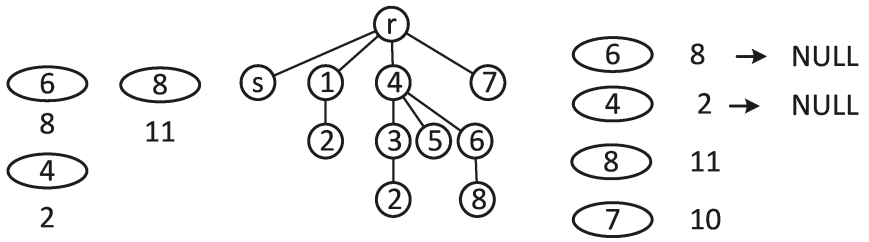

Figure 10. *Step 7*

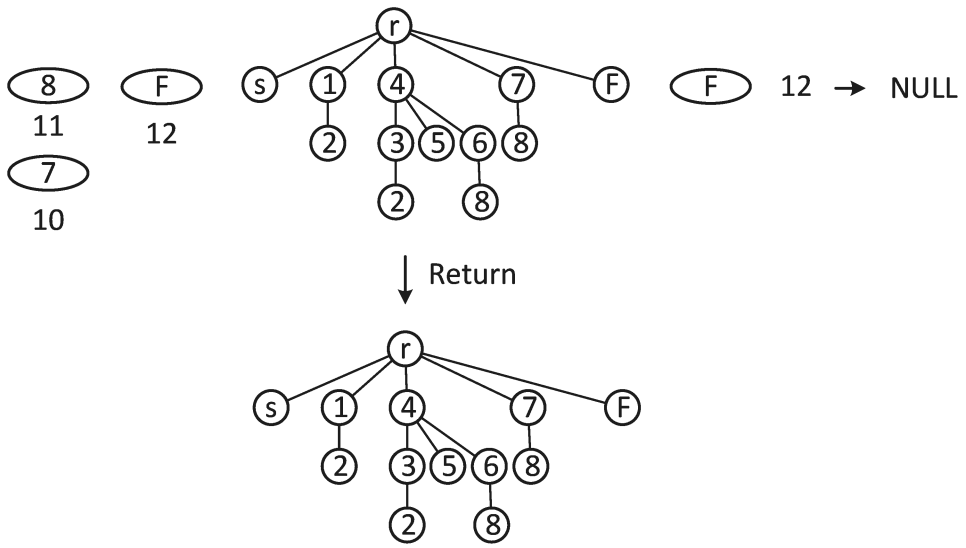

Figure 11. *Steps 8, 9 and finish (10) step*

5. The Discussion of Results. Suggested algorithm based on depth first search. We believe algorithm can be adapted for breadth first search without major changes, because of it is based on operations with active list of routines and corresponding tree of (maximum) cuts while traversing network.

6. Conclusions. The algorithm suggested shall be further improved by its formalization. Research of algorithm complexity shall be provided and tested. We suppose that the network can be restored based on its tree of cuts, and an appropriate algorithm to built the network based on the tree of cuts is possible to create in future.

REFERENCES

[1] M. Carmona and L. Sieh, *Measuring quality in planning: Managing the performance process / Matthew Carmona and Louie Sieh*, Spon, London, 2004.

[2] D.-Z. Cheng, H. Qi, and Z. Li, *Analysis and control of boolean networks: A semi-tensor product approach / Daizhan Cheng, Hongsheng Qi, Zhiqiang Li*, Communications and control engineering, Springer, London, 2011.

[3] A. Geyda and I. Lysenko, *Modeling of Information Operations Effects: Technological Systems Example*, Future Internet, 11 (2019), p. 62, https://doi.org/10.3390/fi11030062.
[4] D. Golenko-Ginzburg and A. Gonik, *Project Planning and Control by Stochastic Network Models*, in Managing and Modelling Complex Projects, T. M. Williams, ed., Springer Netherlands, Dordrecht, 1997, pp. 21–45, https://doi.org/10.1007/978-94-009-0061-5{_}4.
[5] A. Heifetz, *Moves of nature*, in Game Theory, A. Heifetz, ed., Cambridge University Press, Cambridge, 2012, pp. 366–382, https://doi.org/10.1017/CBO9781139049344.029.
[6] B. Hrúz and M. Zhou, *Modeling and control of discrete-event dynamic systems: With Petri nets and other tool / B. Hrúz and M. C. Zhou*, Advanced textbooks in control and signal processing, Springer, Berlin and London, 2007.
[7] T. Liu, Z.-P. Jiang, and D. J. Hill, *Nonlinear control of dynamic networks*, Automation and control engineering, CRC Press/Taylor & Francis Group, Boca Raton, 2014.
[8] J. Melcher, *Process measurement in business process management: Theoretical framework and analysis of several aspects*, KIT Scientific Publishing, Karlsruhe, 2012.
[9] B. Roffel and B. H. Betlem, *Process dynamics and control: Modeling for control and prediction / Brian Roffel and Ben Betlem*, John Wiley, Chichester, 2006.
[10] J. van Bree, *Game based organization design: New tools for complex organizational systems*, Palgrave Macmillan, [Basingstoke], 2013.
[11] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.